Exploring Issues in Set Based Partitioning from Last-level Cache

Rahul Verma
M.tech CSE, IIT Ropar
Ropar, Punjab
2021csm1004@iitrpr.ac.in

Hanumat Lal Vishwakarma
M.tech CSE, IIT Ropar
Ropar, Punjab
2021csm1019@iitrpr.ac.in

***Index Terms*—Set based partitioning, Cache replacement policy, Eviction policy**

I. Introduction

Cache Memory holds a central position in the concept of memory hierarchy to reduce the overhead of frequently accessing the data from the main memory by exploiting temporal and spatial locality thereby saving many CPU cycles. There are various levels of cache memory in the system like L1, L2 and L3(Last level cache) in three level of cache design. Partitioning is one of the important technique in multi-core system. In our discussion, we will be talking about the partition in the Last level cache(L3 in three level cache). Partitioning provides a separate workspace for applications running on each core. After the partitioning one application can only evict their own blocks from LLC(Last level Cache), however in without partitioning any application was evicting the block of other application which was not a good idea considering various aspects of performance and security of the system.

Set based partitioning and Way based partitioning are the two type of partitioning. Way based partitioning are the traditional ways of partitioning, UCP(Utility based cache partitioning) [3] is one such example of way based partitioning. Due to the various limitations in the traditional system of way partitioning, people are shifting to set-based partitioning. We have done various experiments by partitioned(both static and dynamic) our LLC on set based partitioning and collected some of the interesting issues or challenges which developer can face in future(detailed discussion is in Section III).

Further report is divided into five sections, Section II. Related Works & Motivation, Section III. Different issues in set based partitioning, Section IV. Simulation Environments , Section V. Conclusion & Future Works, Section VI. References

II. Related Works and Motivation

In the existing works, researchers have mostly explored and used the way-based partitioning in their various proposals in the multi-core systems. UCP [3] is one such dynamic way-based partitioning algorithm which is based on utility of the application running on each core. In Bespoke [4], there is clearly explained the three possible limitations of the way-based partitioning which are as follows, (1)Scalibility. (2)Security. (3)Fine-Grained Allocations and Flexibility. Further we can also see many issues with way-partitioning in these papers [1]–[4].

Our main motivation behind exploring set-based partitioning was to avoid the scalibility issue of the way-based partitioning. Way-partitioning schemes allocate cache-space at way granularity, with each domain getting one or more LLC ways. But this has limited scalability as the number of partitions is restricted by the cache associativity: e.g., a 16-way cache only supports up to 16 partitions which is inadequate if the number of cores (or threads) increases beyond 16. Even if the number of ways equals number of desired partitions, such a design provides 1-way (direct-mapped) partitions with severe conflict-misses. Lastly, such solutions only provide coarse-grained allocations (e.g., a 32MB 16-way cache provides 2MB partitions), which is inefficient for small working-set programs (e.g., AES).

III. Different Issues got in Set-based Partitioning

As future era is of set-based partitioning. It is very important to explore the various possible future issues/challenges in the Set based partitioning that a developer can face. We found three basic issues in the set-based partitioning which are as follows.

Issue(1):Load imbalance:- Let the example of 2-core system that is there is two partition in LLC. As in set-based partitioning ,We are forcing the mapping(incoming address to LLC set) to appropriate partition as per core number i.e., suppose 2048 sets in partition1(set number:0-2047) and 2048 sets in partition2(set number:2048-4095). And if request of core 0 came into LLC then we have to forcely map to partition1 (because we have fixed this partition for core0) even if, this was mapping to outside the partition1 in without partition.Similarly ,if request of core 1 came into LLC then we have to forcely map to partition2 (because we have fixed this partition for core1) even if, this was mapping to outside the partition2 in without partition. So, Due to this diversion of mapping ,for a range of sets there is high number of evictions from the LLC and for the remaining sets ,number of evictions from sets are less and uniformly distributed. Reason for this is the mapping policy itself which is being used for set-based partitioning.Therefore ,we highly need of the load imbalancing. If we balance the load ,mapping and evictions will be uniformly distributed throught the sets in the LLC. This would definitely reduce the evictions by mapping to some invalid blocks of lesser loaded set.

From the observations, two more issues are possible in the set-based partitioning.

Issue(2):Considerable Invalid blocks in partition:- When we fixed the partition(set-based) then definitely that partition will only be accessed by that core(application) only. Not all the partition is fully utilized i.e., there could be considerable invalid blocks possible in a partition. In worst case every partition having lots of invalid blocks. Suppose a condition where one partition having some invalid blocks and other partition(all ways of all set of partition is full) is full.For every new request of other core there is definitely evictions from it's partition. Even there is invalid blocks in partition1 i.e, empty blocks in partition1 ,other core is evicting it's block. It means if some how we can utilize the invalid blocks of first partition then we can reduce the number of evictions from LLC which will improve the system performance.

Issue(3):Write-back issue when change in partition(set-based) dynamically:- When partition changes dynamically then there are many sets of a partition will fall in other partition i.e., many blocks we need to write back into main-memory after every change in partition. And we know there is latency overhead to write back many blocks after each partition change(dynamically). Therefore, there is latency overhead in every dynamic change of partition. And this could be the serious problem in terms of system performance.

IV. Simulation Environment

We used trace based ChampSim simulator with several parameter configurations to test our results.

Trace1 = gcc_13B.trace.xz

Trace 2 = gcc_56B.trace.xz

Number of cores = 2

Warm-up instructions = 50 millions

Execution instructions =100 millions

LLC Sets = 4096

LLC Ways = 16

Cache Block Size = 64Bytes

V. Experimental Results

As shown in Fig.1, firstly, we did static based partitioning on 2-core system and we counted number of evictions in each set. Experiment is as follows; For baseline comparision we collected number of evictions without applying any partition on LLC.Then we did the partitioning for six different cases. Case 1: we divided the LLC sets in fraction of 1/2, 1/2 for each partition and collected the number of evictions for each set respectively. Case 2: we divided the LLC sets in fraction of 1/3, 2/3 for each partition and collected the number of evictions for each set respectively. Case 3: we divided the LLC sets in fraction of 1/4, 3/4 for each partition and collected the number of evictions for each set respectively. Case 4: we

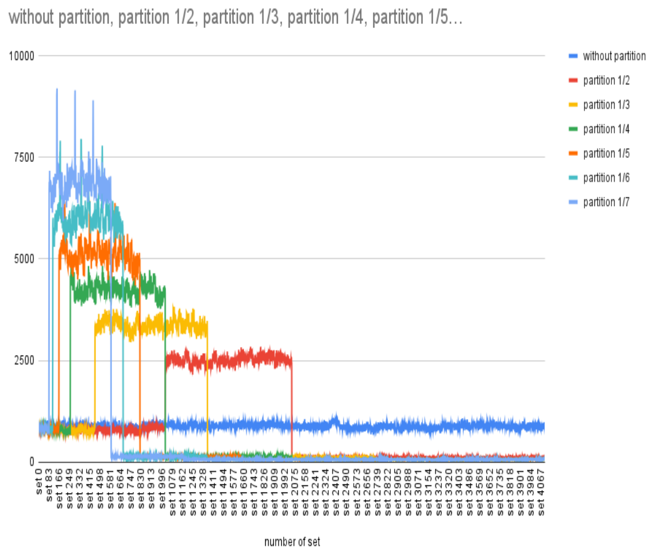

Fig. 1. Number of evictions for each set in static set-based partioning

divided the LLC sets in fraction of 1/5, 4/5 for each partition and collected the number of evictions for each set respectively. Case 5: we divided the LLC sets in fraction of 1/6, 5/6 for each partition and collected the number of evictions for each set respectively. Case 6: we divided the LLC sets in fraction of 1/7, 6/7 for each partition and collected the number of evictions for each set respectively. After that we plot the graph for the number of eviction vs number of sets on changing the partitioning size statically.

For a range of sets we were getting high number of evictions while for the remaining sets there is uniform distribution in number of evictions, which is due to the strict mapping policy of the set based partitioning. From the comparision of the different cases and baseline it is highly need to do the load balancing.

As shown in Fig.2, we did dynamic based partitioning on 2-core system and we counted number of evictions in each set. Experiment is as follows; we changed the partition dynamically based on number of instructions. Case 1 is for when number of instructions are below 15M (by dividing the LLC sets in fraction of 1/2, 1/2), Case 2 is when number of instructions are above 15M and below 30M(by dividing the LLC sets in fraction of 1/3, 2/3), Case 3 is when number of instructions are above 60M and below 75M(by dividing the LLC sets in fraction of 1/4, 3/4), Case 4 is when number of instructions are above 75M and below 90M(by dividing the LLC sets in fraction of 1/5, 4/5) and Case 5 is for remaining instructions(by dividing the LLC sets in fraction of 1/6, 5/6). After that we plot the graph for the number of eviction vs number of sets on changing the partitioning size dynamically.

Similarly, we got similar results here also as in static set based partitoning i.e.,For a range of sets we were getting high number of evictions while for the remaining sets there is

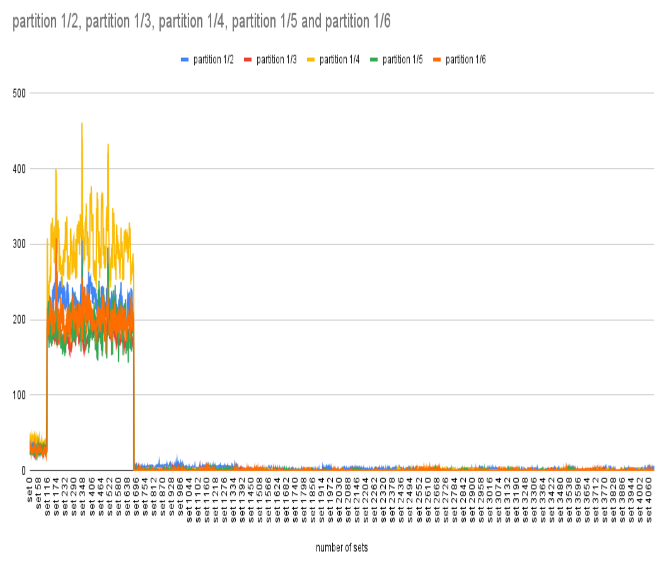

Fig. 2. Number of evictions for each set in dynamic set-based partioning

uniform distribution in number of evictions, which is due to the strict mapping policy of the set based partitioning. From the comparision of the different cases and baseline it is highly need to do the load balancing.

VI. Conclusion and Future Works

Finally we can conclude that existing partitioning schemes relying on way-partitioning face scalability and flexibility challenges that limit their adoption. So, we explored set-partitioning but there are many possible issues in it like we discussed above, which is needed to be addressed that can impact the system performance. Therefore, while choosing set based partitioning scheme for multi-core system we must consider these issues in mind.

References

[1] Samira Manabi Khan, Yingying Tian, and Daniel A Jimenez. Sampling dead block prediction for last-level caches. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 175–186. IEEE, 2010.

[2] Moinuddin K Qureshi, Aamer Jaleel, Yale N Patt, Simon C Steely, and Joel Emer. Adaptive insertion policies for high performance caching. *ACM SIGARCH Computer Architecture News*, 35(2):381–391, 2007.

[3] Moinuddin K Qureshi and Yale N Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 423–432. IEEE, 2006.

[4] Gururaj Saileshwar, Sanjay Kariyappa, and Moinuddin Qureshi. Bespoke cache enclaves: Fine-grained and scalable isolation from cache side-channels via flexible set-partitioning. In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, pages 37–49. IEEE, 2021.